More Detail

The Built in Assembler

For the full gory details, see http://www.nongnu.org/avr-libc/user-manual/assembler.html

and the 'cookbook' at http://www.nongnu.org/avr-libc/user-manual/inline_asm.html

It is *very* rare to need to use the assembler. Normally the requirement is to do a single `no-op` without changing anything, in order to delay by 1 CPU cycle. This can be achieved with:

```
asm("and r0,r0");
```

The Makefile

A simple description of make and Makefiles is available here: http://www.network-theory.co.uk/docs/gccintro/gccintro_16.html

That covers everything in the sample Makefile except the first few lines:

`CPPFLAGS=-I. -I../lib` instructs the compiler to add two -I (=Include) flags.

-I. instructs the compiler to look for .h (header) files in the current directory.

-I../lib instructs the compiler to also look in the lib directory at ../lib

`MCU=atmega168p` is required by gcc-avr to tell it which device is the target of the compilation. The list of supported devices is here: http://www.nongnu.org/avr-libc/user-manual

`VPATH=../lib` make will look for source files in the current directory, then in directories listed after `VPATH=`.

Libraries

For the details, see http://www.nongnu.org/avr-libc/user-manual/library.html

Common pitfalls

Here is a list of common pitfalls, in no particular order.

Interrupts

One of the easiest traps to fall into is to spend so long in the interrupt routine, that another interrupt has occurred before the first one is finished. If timing is critcal, then the extra latency before the second interrupt is serviced will probably cause a failure. This suggests doing the minimum possible in the interrupt service routine (ISR), eg setting flags, and the maximum outside the ISR.

A different strategy to the above is as follows:

Do everything in the interrupt routines, and nothing in main. Make sure that each interrupt handler sets up the conditions for the next interrupt to occur. If only one interrupt is enabled at a time, then there is no potential to clash.

Make sure that all interrupts are set up and conditions cleared before the Master Interrupt Enable bit is set, otherwise an interrupt may occur immediately.

If you use global variables which are changed in more than one place, declare such variables as `volatile`, otherwise the compiler will optimise the code, and not notice that the variable has changed.

I/O

It is very easy when modifying code to forget to define a new output. Everything will compile correctly, but the pin will not behave as you expect.

It is useful to copy a line such as `PORTB |= (1<<PB1)`, changing it to `PORTB &= ~(1<<PB1)`. However, omitting the brackets in the second case will cause problems because of operator precedence.

Except while initialising, it is *very* rare that you will need assignment such as `PORTB = (1<<PB2);` You almost always need `|=` or `&=`

Analogue to Digital converter

The most common pitfall with the A to D is forgetting to supply a reference volatge at AVcc. Zero or very low readings are a symptom of this.

The next most common is having too high a clock rate. If it is too fast then only the first few bits of the data will be correct.

Selecting the wrong analogue input is the next most common error.

Fuse errors

If you can no longer program the fuses, have you disabled the clock by mistake ? This is particularly easy to do when chaning from an internal to an external or crystal derived clock.

If you reprogram the fuses, the device does not automatically restart. You must power cycle it. Changing the program *does* however force a restart.